

IMAGE SECURITY USING INTRA-FILE SECURITY

Sura N. Abdulla

Department of Computer Science, College of Science, University of Baghdad. Baghdad-Iraq.

Abstract

Typical image cryptographic systems provide security by encrypting entire image. This has the advantage of simplicity, but does not allow for fine-grained protection of data within very large image files, which is very important in some applications where some but not all the image is sensitive or classified. In this paper, a new method is proposed for securing parts of the image (i.e. sensitive information). The proposed method combines the Intra-File security, image inpainting and cryptography techniques to secure the sensitive parts of an image and leave the rest of the image without any changes, further more, it allows multiple levels of security for the same image without the need to have more than one copy of that image. Experimental results of the proposed method provides better security than the classical methods.

الخلاصة

إن الطرق التقليدية المستخدمة لتشفير الصور تزودنا بالحماية لهذه الصور عن طريق تشفيرها كاملة، و هذه الطرق تجعل عملية التشفير تكون سهلة لكنها لا تسمح بتلبية بعض المتطلبات للمستخدمين التي تحتاج لحماية بعض المعلومات الموجودة في الصورة مع ترك باقي الصورة بدون تشفير خصوصا إذا كانت الصورة كبيرة جدا والمعلومات التي نحتاج لحمايتها صغيرة جدا مقارنة بحجم الصورة. يقدم هذا البحث طريقة جديدة لإخفاء أجزاء من الصورة ضمن نفس الملف الذي يحوي الصورة الأصلية حيث استخدمنا طريقة معروفة للحماية تسمى Intra-File Security لغرض إخفاء تلك الأجزاء واستخدمنا كذلك الطريقة المسماة Inpainting لغرض تعويض الأجزاء المخفية من الصورة بحيث تبدو الصورة للناظر لها متكاملة ولا يوجد فيها شيء ناقص. كذلك استعنا بالطرق المعروفة في التشفير لغرض تشفير الأجزاء المخفية من الصورة قبل إخفائها في نفس الملف. أوضحت نتيجة تجربة هذه الطريقة إنها توفر حماية و أمنية جيدة مقارنة بالطرق التقليدية المستخدمة لحماية الصور.

Introduction

During this time when the Internet provides essential communication between tens of millions of people and is being increasingly used as a tool for commerce, security becomes a tremendously important issue to deal with. A secure system should still permit authorized user to carry out legitimate and useful tasks. One might be able to secure a computer beyond misuse using extreme measures:- "The only truly secure system is one that is powered off, cast in a block of concrete and sealed in a lead-lined room with armed guards and even then we

may have doubts" [1]. According to that, a full secured system is something nearly impossible to exist. Instead, we have some methods that can be used to secure the sensitive information. Computer images may contain large amount of information that users often want to keep it confidential and secret such that no body (unauthorized users) can spy it. This information may include technique, commercial, military zones, personal and much more. However, unauthorized users can act in two different ways:- *passive* and *active intruders*. Passive intruders just want to view the image which they

are not authorized to view it. On the other hand, active intruders want to make unauthorized changes to those images [2].

Most of the personal computers today use Windows Operating System. However, Windows does not provide any security level in the files stored on the hard or floppy disk. Files contents can be read, deleted, copied, exchanged and stolen very easily [3].

In order to protect images from unauthorized users we need to secure them against those users. Traditionally, image security uses an *all-or-nothing* approach (all of an image is encrypted identically). This approach is sufficient in situations where an image must be viewed in its entirety to make sense for a user application. However, there are many cases where a user should only have view to part of the image and the rest of it must be hidden from that user. For example, a personal picture stored in a personal computer, that can be accessed by more than one user, may contain sensitive parts that the user wants to hide it from all or some of the other users for a personal purposes. Other example include a satellite map of a region containing military zones, a specification for a vehicle with sensitive information. Users that desire different levels of security must use different copies of that image, complicating access for all users, and using storage space in such an inefficient way specially when we have hundreds of images to be stored.

In some applications, it is relevant to hide content of a message when it enters an insecure channel. Cryptography, steganography and operating system techniques are methods used to protect files from the intruders [2].

Cryptography is the art or science of keeping messages secret and ensuring that messages cannot be easily read or modified by unauthorized users [3]. The initial message prepared by the sender is then converted into ciphertext prior to transmission. The process of converting plaintext into ciphertext is called *Encryption*. The process of recovering plaintext from ciphertext is called *Decryption* [4][5][6] (see Figure 1).

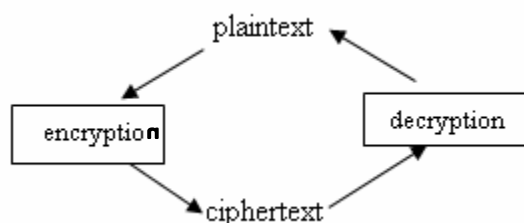


Figure (1): The encryption/ decryption process

This paper is organized as follows: Section 2 illustrates digital image inpainting, Section 3 illustrates Intra-File Security, Section 4 describes the proposed algorithm, Section 5 is for the results, and Section 6 is for the conclusions.

Inpainting

The restoration of digital images can be carried out using two salient approaches [7]:

- 1- Image inpainting techniques for filling in small image gabs.
- 2- Texture synthesis algorithms for generating large images regions from sample textures.

Image inpainting [8] [9] is the process of filling in missing data in a designated region of a still or video image, it is a practice that predates the age of computers. Image inpainting is a technique used by art museum crafts men to fill in parts of a painting which have decayed or been damaged over the course of many years. It is called inpainting as a literal terms for the process of painting in holes or cracks. Digital inpainting refers to inpainting of digitized images with computer support[10]. The notion of digital inpainting was first introduced in the year 2000 by the Bertalmio-Sapiro-Caselles-Ballester paper [9]. Digital image inpainting has received attention over the few years, because of the many applications that is in image processing, including removal of scratches, objects, text or logos from digital images to re-touching damaged paintings and photographs [7] [9], it can also be used for *airbrushing* to remove unwanted image details[10].

The problem in inpainting is passed as follows: given an area to be inpainted, filling in the missing areas or modifying the damaged ones in a non-detectable way for an observer not familiar with the original images[7]; in other words, the problem of local inpainting is[9][11][12]: let I_0 be the original image and I be the inpainted image given a region to be inpainted Ω in I_0 and its boundary $\partial\Omega$, we must synthesize pixel values from the boundary inwards, using neighborhood pixel information to continue the inpainting process.

The following pseudo-code describes the general solution to the problem:

- 1- Specify the region to be inpainted Ω .
- 2- Compute the boundary $\partial\Omega$ of the region Ω .
- 3- Initialize Ω by clearing existing color information.
- 4- Repeat for all pixels $p(x,y)$ in Ω :
 - 4.1- Inpaint $p(x,y)$ in Ω based on information from its surrounding pixels.

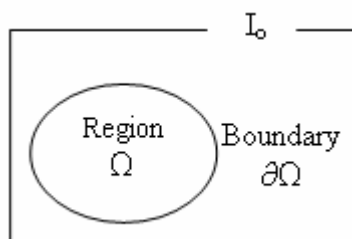


Figure (2): local inpainting.

We assume a prior information about the probability distribution of the relation between a pixel value and its neighborhood, which will help fill in a pixel lying on the hole boundary.

On the other hand, texture synthesis accepts a given sample texture and creates an output image which can have arbitrary dimensions, but which retains the texture properties derived from the original sample[7]. We can think of the texture synthesis problem, which requires an input texture, as reducing to the inpainting problem if we assume that the sample or input texture which it attempts to replicate can be found in the same image where the region to be synthesized lies[7].

Intra-File Security (IFS)

Cryptographic file systems typically provide security by encryption entire files or directories. This has the advantages of simplicity and it is sufficient in situations where a file must be accessed in its entirety to make sense for a user or application, but it does not allow for fine-grained protection of data within very large files. This is not an issue in most generated-purpose systems, but can be very important in scientific applications where some but not all of the entire output data is sensitive or classified. However, there are many applications that would benefit from the ability to encrypt data in smaller pieces using different keys to permit parts of a file to be read and write by different groups of users, and there are many cases where a user should only have access to some of the data in a file such as a large file used for scientific modeling might contain mostly unclassified information with some sections of classified data. Using old techniques, user that desire different levels of security must use different file, which means that we should have a secured file for every group of users, and that yields to complicating access for all users beside the large storage place that is taken for storing all the secured files.

Using Intra-File encryption approach gives us the ability to use common cryptographic techniques to secure any arbitrary-sized region of data within a file, even if the region is logically noncontiguous[13][2], it allows mixing data of different sensitivity in a single file. This benefits users by permitting related data belonging to a single file to be kept together rather than separating data of different security needs. IFS allows users to encrypt extents of files independently from other extents, so that a single file may contain one or more secure regions. A file system incorporating IFS transparently handles most operations, such as automatic decryption and key management. The result is a file system with little extra programming or runtime overhead for the added functionality [13].

IFS uses additional metadata to maintain information about secure segments, allowing blocks of a file to be encrypted and decrypted individually on the client.

IFS allows encryption to be applied to segments as small as a byte or as large as an entire file, multiple encrypted segments need not be logically contiguous with in the file. In an IFS file, encrypted data is stored logically in-place, and occupied the physical file blocks that would have contained the unencrypted data [13]. To support efficient random file access, we independently encrypt data from each logical file block, so there is no dependence on information from other blocks.

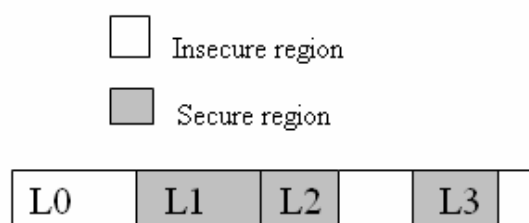


Figure (3): A single logical file address space broken into secure and insecure regions.

Consider the file shown in Figure 3, which contains a non-contiguous region that must be kept secure, the region spans are entire logical block (L1), and two partial blocks (L2 and L3), this region is not independently encryptable using standard techniques. With IFS, this non-contiguous region of the file can be encrypted independently and made available only to appropriate users. Furthermore, because the

encrypted data is left in place, all programs written to work with the full data set can still function properly. All regions of the data encrypted and unencrypted alike, will still be readable except that the encrypted regions will not contain the secured data values but will contain apparently random values [13].

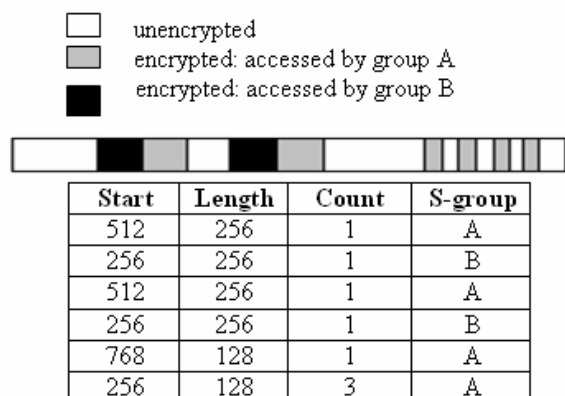


Figure (4): A 4 KB block encrypted with intra-file security and its associated security node (s-node). Note that the last entry in the s-node has a repeat count of 3, representing the three repeated secure regions near the end of the file. The first of the four regions must be represented separately because its distance from the previous region is larger than that of the following three regions.

By default, all data in the file assumed to be unencrypted. In order to locate the secure data within the file, and to find the encryption parameters, each encrypted block requires a description of the location of secure segments and initialization vector information. In IFS, the structure holding this data is a *security node*, as shown in Figure 4. Notice that each security node consists of four information: the *Start* information which is relative to the start of the previous secure region, or the start of the block for the first region. The *Length* information shows the length of the region in bytes. Some times many secure regions are formed of repeating patterns of data of varying levels of security, so the table contain a shorthand way of representing simple patterns of data of secure regions that are a fixed length and fixed distance apart. This is accomplished by specifying a repetition *Count* associated with the offset and length specified in the secure region specification.

The last information in each node is *S-group*, which store the information necessary to encrypt and decrypt the secured data and includes key information for the region as well as an initialization vector (IV) – a number used to

seed the encryption algorithm when it operates on the encrypted data in the block. The IV is a function of the logical block number as well as per-file values such as file identifier, that's why it need not stored in the security-node because it can be calculated at runtime. The IV is necessary to ensure that encrypted regions with the same data do not result in the same ciphertext. We need also to store pointers to keys in that security-node and avoid storing key information in the security-node, and since we have more than one key then we have more than one security group, that's why we store *S-group* identifier for each secure region, this identifier is translated by the system into a key.

The security-node as depicted in Figure 4 is simple to implement, but uses space inefficiently [13]. Instead, security-nodes could be compressed using any technique for compressing small numbers.

The Proposed Algorithm

The common requirement for hole filling algorithms is that the region to be filled has to be defined by the user. This is because the definition of a hole or distortion in the image is largely perceptual. It is impossible to mathematically define what a hole in an image is [7]. A fully automatically hole filling tool may not just be unfeasible but also undesirable, because in some cases, it is likely it remove features of the image which are actually desired. The user should have control over the selection of the region after which the filling is entirely automatic across the spectrum, of methods to fill texture or inpainting[7].

In this section, we describe an algorithm that can be used for securing different regions on one image. The steps of the proposed algorithm can simply be stated as follows:

Step 1: read the original image.

Step 2: repeat the following steps for each desired sensitive region:

- 1- Select rectangular area starting at pixel $p(x_1, y_1)$ and ending at pixel $p(x_2, y_2)$, this area will be the desired region Ω .
- 2- Store each pixel $p(x, y)$ from region Ω in temporary matrix.
- 3- Change the colors of all pixels in region Ω to be one color.
- 4- Inpaint the pixels of region Ω to fill-in the gab that we made in the last step .
- 5- In the Intra-File table, store the following information: the coordinates of region Ω , a number representing the group of users

allowed to view region Ω , and number of pixels in this region.

Step 3: Store all temporary matrices in a one dimensional vector.

Step 4: Store the new inpainted image in a BMP file.

Step 5: Store the Intra-File table and the one dimensional vector in the BMP file(at the end of the new image).

Notice that, in step 2 we needed a rectangular shape to be the region Ω , since this is the best shape for our work because we only need to store x_1, y_1, x_2 and y_2 (which represents the top left and the bottom right corners of the rectangular region) to determine the boundary of region Ω , that will give us the ability to retrieve the original image later without the need for any extra calculations.

For inpainting the selected region we used algorithm called "Evaluation inpainting method" which utilizes the search capabilities of Evolutionary Algorithms (EA) for finding the appropriate pixels to inpaint the selected region. The algorithm automatically fills-in the selected region Ω from the promising EA pixels around that region. The region to be inpainted must be selected by the user depending on his subjective selection. The user indicates the region Ω in an image I_o , to be selected. This step creates a mask that covers the selected region completely. Suppose that $p(x,y)$ represents a pixel in the selected region Ω at coordinates x and y , and $I_o(x_o, y_o)$ represents any pixel in the image I_o with the coordinates x_o and y_o excluding Ω . Moreover, let us assume that there is a rectangular region $\partial\Omega$ surrounds the masked region Ω with some preselected width B_w . At this point, the only user involvement to the algorithm requirements is to mask the region Ω to be inpainted. The algorithm is used iteratively to fill-in the selected region in raster scan order, from top to bottom and from left to right as follows: for each pixel $p(x,y)$ in Ω (starting from the pixel in the top left corner of Ω), create randomly an initial population consisting of six groups of individuals selected from $\partial\Omega$ region(as shown in Figure 5). The first and the second groups of individuals are chosen from the left- and right-sides of rectangular region passing the horizontal line with the pixel $p(x,y)$ respectively. The third and fourth groups of individuals are chosen from top- and bottom-sides of a rectangular region passing the vertical line with $p(x,y)$ to be inpainted. The reason for these four groups selection can be traced back to

the fact that these regions may contain promising solutions to the current pixel $p(x,y)$ to be inpainted. The fifth group is chosen randomly from $\partial\Omega$, depending on the boundary Ω . The sixth group is been calculated from the previously inpainted pixel. Then compute the fitness value of each selected group, executing and repeating calculations on the pixels of each selected group to find the pixels with the best fitness value. The last step is to update the value of the current pixel $p(x,y)$ inside he region Ω to be filled with the fittest pixel from the selected groups(This is a brief explanation for the EA algorithm, for more details refer to reference [12]). This algorithm works good with small regions and with large regions too, which is good for our work.

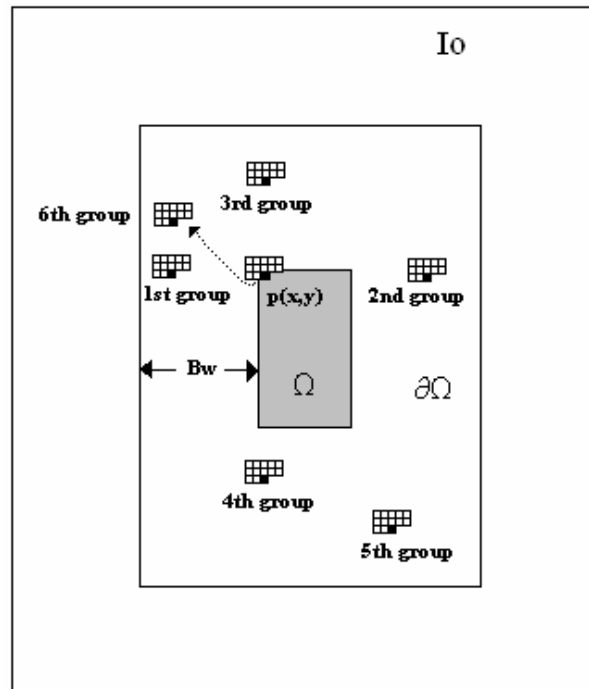


Figure (5): initialization procedure

We have to decide from the beginning the groups of users allowed to view the image, each group can view only parts of the image, and choose encryption method for each users-group. Users who are allowed to view the sensitive parts of the image are grouped together in separated groups, each group of users is given a number (s-group) to be used later as an identification to them, which means: group no.1 are allowed to view the first region Ω only, group no.2 are allowed to view the second region Ω only, and so on. Each users group will have a different encryption method. The complexity of that encryption method depends

upon the degree of sensitivity or importance of the corresponding hidden region Ω , if the hidden region Ω is very sensitive region and it must be very hard to be viewed by other users groups or any unauthorized users, then the technique used to encrypt it must be complex and hardly to be broken by intruders. We need one encryption method for each sensitive region, then the number of encryption methods needed for one image is equal to number of sensitive regions in that image, which is equal to number of s-groups that we had.

Each security node in our Intra-File table has the following fields:

Field name	purpose
Start_X	The X axis of the top left corner of region Ω .
Start_Y	The Y axis of the top left corner of region Ω .
End_X	The X axis of the bottom right corner of region Ω .
End_Y	The Y axis of the bottom right corner of region Ω .
length	Number of pixels in the temporary matrix of region Ω .
s_group	Number represent the group of users allowed to view region Ω , used also to determine the encryption method for the region.

Each security node in the Intra-File table contains information about one hidden region Ω in the original image file. We store this table at the end of the inpainted image, in the same BMP file.

we can't leave the encrypted regions in its original place in the image, because any viewer for that image will notice those regions immediately, that's why we replace them with inpainted regions, then we need to keep them in one dimensional vector that holds all the encrypted regions one after the other, and then stored that vector after the Intra-File table.

Each software usually used to open BMP, reads the header table first then displays the image according to the information stored in those tables, and since we stored the Intra-File table at the end of the BMP file without changing those fields then any software opens that BMP file will ignore the extra information that we added and the user will never notice the hidden information.

Results

In Figure 6 we show the original image, image with selected regions for hiding them, and the resulted image that contains the inpainted image with the hidden Intra-File and the one dimensional vector.

Using the proposed algorithm for hiding only the sensitive parts of an image will give a good security for the desired image according to the following:

- 1- When an intruder views the secured image he will see an ordinary picture without any gaps in it, each hidden part is inpainted in such away that makes it hard to determine which parts of the image are inpainted.
- 2- The size of the secured image file is an ordinary size, it does not bring any suspicions because its not a large size that can be noticed by any intruder.
- 3- The original image file is hidden no one can reach it, only the secured image file is available for every body. Hence, no one can have the ability to make a comparison between the sizes of the two image files, which means that no one can have the ability to notice the little change in the size of the secured image file.
- 4- Using the available software (such as Photoshop) to view the secured image will show only the secured inpainted image without giving any notice or hint about the hidden data at the end of the file, because it depends on the information stored in the BMP file header to determine the dimensions of the image. The proposed algorithm stores the hidden data at the end of the new BMP file and uses the header of the original image to be a header for the new secured image, which means that the secured image will have the same dimensions of the original image .
- 5- Reading data from a BMP file needs a special care [14][15], it needs a previous knowledge about how that data is stored otherwise the information read from that file will be all wrong and worthless.
- 6- The hidden information is stored at the end of the file as a sequence of numbers, a sequence of encrypted data where several encryption methods were used, one for each group of users. Only the authorized group of users have the ability to find and decrypt the information that corresponds to the region which they have its decryption key to view it.










Original image	Image with selected regions ready to be hidden	Inpainted image with hidden Intra-File and one dimensional vector
<p>(1)</p>  <p>Dimensions: 180 X 135</p>		 <p>Dimensions: 180 X 135</p>
<p>(2)</p>  <p>Dimensions: 200 X 263</p>		 <p>Dimensions: 200 X 263</p>
<p>(3)</p>  <p>Dimensions: 444 X 636</p>		 <p>Dimensions: 444 X 636</p>

Figure (6): Images before and after using Image Security Using Intra-File Security

Conclusions

In this paper, we had introduced a new method that uses Intra-File tables for hiding sensitive parts of images inside the same image file. If any unauthorized user try to open the secured image using the available software (e.g. windows paint software, ACDsee software, Photoshop software, ...) he will get the inpainted image only without noticing that there are hidden information after that image and that he needs a special program to reach the hidden Intra-File table and the secured parts of that image.

References

1. Wikipedia , February **2007**, "*computer security*", available at: http://en.wikipedia.org/wiki/computer_security/
2. Al-Barak A., **2004**, "*Intra-File Security System*", A thesis submitted to Computer Science Department, College of Science, University of Baghdad.
3. Rafal Swiecki, **2004**, "*Introduction to cryptography-principles and systems: principles of cryptography*", available at: <http://www.minelinks.com/supercode/index.html>
4. Oli Cooper, **2001**, "*An introduction to Cryptography*" , available at: <http://www.cs.bris.ac.uk/~cooper/cryptography/crypto.html>
5. Mare Van Droogenbroeck and Raphael Benedett , September **2002** , "*techniques for a selective encryption of uncompressed and compressed imager*" , ACIVS Advanced Concepts for intelligent Vision Systems, Ghent, Belgium, pages 90-97.
6. Gary C. Kessler, (February 2006), "An Overview of Cryptography", available at: <http://mio.ece.uic.edu/~papers/WWW/cryptography.html> .
7. Nined Pradhan , **2004** , "*Digital Image Restoration Techniques and Automation*", available at : www.ces.clemson.edu/~stb/ece847/fall2004/projects/proj06.pdf
8. Bertalimo M., Bertozzi A.L., Sapiro G., December **2001** , "*Navier-Stokes, Fluid Dynamics and Image and Vidio Inpainting*" , . Proc. IEEE Computer Vision and Pattern Recognition (CVPR'01) , Hawaii.
9. Bartimio M., Sapiro G., Ballester C. and Caselles V., July **2000**, "*Image Inpainting*" , "*Computer Graphics*", SIGGRAPH 2000,pp. 417-424.
10. Harald Grossauer , "*Digital Inpainting*" . Department of Computer Science. University Innsbruck, A-6020 Innsbruck, Austria. Available at: http://www.it.lut.film.at/EcmiNL/ecmi34/nod_e6.htm.
11. Olivera M. M., Bowen B., McKenna R., Chang Y., September **2001**, "*Fast Digital Image Inpainting*", Imaging and Image Processing (VIIP 2001)
12. Al-Robaie Z., November **2005**, "*Inpainting Problem Based on Evolutionary Algorithms*", A thesis submitted to Computer Science Department, College of Science, University of Baghdad..
13. Scott A. Banachowski , Z. N. J. Peterson, E. L. Miller & Brandt S. A., "*Intra-File Security for a Distributed File System*", available at: ssrc.cse.edu/papers/banachousski-mss02.pdf
14. Stefan Hatzl, **1998**, "*The .bmp file format*", available at: <http://www.fortunecity.com/skyscraper/windows/364/bmpffrmt.html>
15. UCCS, **2004**, "*Windows BMP Bitmap File Format*", available at: [http://web.uccs.edu/wbahn/ece1021/STATIC/REFERENCE S/bmpfileformat.htm](http://web.uccs.edu/wbahn/ece1021/STATIC/REFERENCE/S/bmpfileformat.htm)